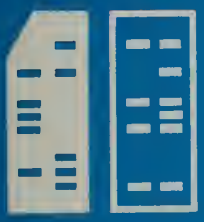
Report No. 310

NUCLEOL AS A FORMAL SYSTEM

by

Marek Pawel Irwin-Zarecki

February 21, 1969



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 310

NUCLEOL AS A FORMAL SYSTEM

by

Marek Pawel Irwin-Zarecki

February 21, 1969

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the Department of Computer Science at the University of Illinois at Urbana-Champaign, Urbana, Illinois, the National Science Foundation under NSF Grant GJ-217, by BUILD, the project which is sponsoring cooperation on EOL-4 between the University of Illinois and the University of Colorado and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, February 1969.

ACKNOWLEDGMENT

I would like to express my appreciation to my advisor, Dr. Jurg Nievergelt, Associate Professor of Computer Science and Mathematics, for his guidance and supervision of this thesis. I also gratefully acknowledge the assistance of Dr. Leon Lukaszewicz, Visiting Professor of Computer Science (1966-67) and Director of the Institute of Mathematical Machines, Polish Academy of Science. Also, I wish to express my thanks to Mrs. Freda Fischer, Senior Research Programmer, and Mr. John Sidlo, Graduate Research Assistant of the Department of Computer Science. I would like to acknowledge the personnel of the IBM 360/75 service area of the Department of Computer Science at the University of Illinois at Urbana-Champaign for their assistance.

This work was supported by the Department of Computer Science at the University of Illinois at Urbana-Champaign, the National Science Foundation under NSF Grant GJ217, and by BUILD, the project which is sponsoring cooperation on EOL-4 between the University of Illinois and the University of Colorado.

I would like to express my sincere appreciation to my wife, Basia, for her encouragement during this study.

TABLE OF CONTENTS

1. INTRODUCTION

This thesis is presented as a contribution to the state of the art in formal definition of programming languages.

The ALGOL 60 report (1) started the trend by showing the feasibility of a formal definition of the syntax of programming languages. Since then efforts have mainly been concentrated on a definition of semantics. Notable among the latter is McCarthy's approach (2,3) which has strongly influenced the present paper. However, McCarthy applies his method only to very small parts of programming languages, and it is not clear what the complexity of a formal definition of a full programming language would be.

A research group at the IBM laboratory in Vienna, Austria, has applied techniques which are substantially the same as McCarthy's to the formal definition of PL/1. Their reports, which so far amount to approximately 1000 pages, are summarized in (4).

This paper attempts to present a formal definition of NUCLEOL, a small but useable list processing language. NUCLEOL was designed by Freda Fischer, Marek Irwin-Zarecki, Jurg Nievergelt, and John Sidlo and has been described in Sidlo's M.S. thesis (5). A significant aspect of this thesis is its insistance that a formal definition should define the language both for humans and for computers. Accordingly, the definition has been partially written in PL/1 and it constitutes about a half of the implementation of NUCLEOL.

The plan of this thesis is as follows. In the second part (following this introduction) I define a formal system called NUCLEOL.

I list basic predicates and functions and indicate the axioms that they must satisfy. Next I define the flow of control and the meaning of the NUCLEOL instructions. This definition is written in the form of a PL/1 program. In the third part I present the NUCLEOL programming language as a model for the NUCLEOL formal system. The section 3.1 contains the syntax of the NUCLEOL programming language (somewhat different from (5)). The section 3.2 shows how NUCLEOL functions and predicates are reflected in the NUCLEOL programming language, thus showing that the NUCLEOL programming language is indeed a model, in the sense of logic, for the NUCLEOL formal system. The fourth part is somewhat unrelated to the main topic of this thesis. It contains a macro generator, written in NUCLEOL itself, which allows NUCLEOL to be extended by means of macros. It also serves as an example of a NUCLEOL program.

2. NUCLEOL FORMAL SYSTEM

NUCLEOL FORMAL SYSTEM IS DEFINED TO BE
< ALPHABET , TERMS , FORMULAS > ;

ALPHABET = < VARIABLES , BASIC FUNCTIONS , CONSTANTS ,
BASIC PREDICATES , = , LOGICAL CONNECTIVES ,
AUXILIARY SYMBOLS > ;

THE SET OF VARIABLES INCLUDES:
STATE,WELL_FORMED_STRING,BLOCK,CONSTITUENT,TYPE,DIRECTION,
BITSTRING,CHARACTERSTRING,NUMBER,NAME,WFS_NAME,WFS,C,D,
AND SOME OTHERS SUBSCRIPTED VARIABLES
/* NAMES OF VARIABLES DO NOT HAVE ANY FORMAL MEANING */ ;

THE SET OF BASIC FUNCTIONS IS LISTED IN 2.1. ;

THE SET OF CONSTANTS INCLUDES:
$B , $C , $D , $P , $R ,
$LEFT_PARENT , $RIGHT_PARENT ,
LEFT , RIGHT ;

THE SET OF BASIC PREDICATES IS LISTED IN 2.1. ;

= IS THE EQUALITY RELATION SYMBOL ;

THE SET OF LOGICAL CONNECTIVES INCLUDES:
| /* OR */ ,
& /* AND */ ,
=> /* IF..., THEN... */ ,
<=> /* IF AND ONLY IF */ ,
¬ /* NOT */ ;

THE SET OF AUXILIARY SYMBOLS INCLUDES:
(/* LEFT PARENTHESIS */ ,
) /* RIGHT PARENTHESIS */ ,
, /* COMMA */ ;

THE SET OF TERMS IS DEFINED INDUCTIVELY AS FOLLOWS:
VARIABLE IS A TERM ;
CONSTANT IS A TERM ;

IF FUN IS A BASIC FUNCTION AND T1,T2,...,TN ARE TERMS THEN
FUN(T1,T2,...,TN) IS A TERM ;

THE SET OF FORMULAS IS DEFINED INDUCTIVELY AS FOLLOWS:
IF T1 AND T2 ARE TERMS THEN
T1=T2 IS A FORMULA ;
IF P IS A BASIC PREDICATE AND T1,...,TN ARE TERMS THEN
P(T1,...,TN) IS A FORMULA ;

```
IF FORM1 AND FORM2 ARE FORMULAS THEN
FORM1 | FORM2 ,
FORM1 & FORM2 ,
FORM1 => FORM2 ,
FORM1 <=> FORM2
ARE FORMULAS ;
IF FORM IS A FORMULA THEN
¬FORM IS A FORMULA ;
```

2.1. NUCLEUL BASIC FUNCTIONS AND PREDICATES

```
/* ALL ARGUMENTS AND YIELDED RESULTS OF THE BASIC FUNCTIONS
   AND PREDICATES ARE FURMAL VARIABLES. NAMES UF THE FORMAL
   VARIABLES INFORMALLY INDICATE KINDS OF EXPECTED ARGUMENTS
   AND RESULTS. */

    BASIC PREDICATES :
    CATEGORY PREDICATES :
IS_STATE(STATE) ;
IS_WFS(WELL_FORMED_STRING) ;
IS_BLOCK(BLOCK) ;
IS_CONSTITUENT(CONSTITUENT) ;
IS_TYPE(TYPE) ;
IS_DIRECTION(DIRECTION) ;

IS_BITS(BITSTRING) ;
IS_CHRS(CHARACTERSTRING) ;
IS_NUMB(NUMBER) ;
IS_NAME(NAME) ;

   OTHER PREDICATES :
IS_CONVERTIBLE(CONVERSION_MODE,CONSTITUENT) ;
CAN_PASS(SCANNER_ATTRIBUTE,PARENTHESIS_ATTRIBUTE) ;
TESTS(TEST_MODE,CONSTITUENT1,CONSTITUENT2) ;

    BASIC FUNCTIONS :
    STATE LEVEL FUNCTIONS :
EXEC(STATE)=WFS_NAME ;
CONTENT(WFS_NAME,STATE)=WFS ;
CONTENT_OF_EXEC(STATE)=WFS ;
WFS_NAMED(WFS_NAME)=WFS ;
KILL(WFS_NAME,STATE)=NEW_STATE ;
CREATE(WFS_NAME,WFS,STATE)=NEW_STATE ;
ALTER(WFS_NAME,WFS,STATE)=NEW_STATE ;
MOVES(DIRECTION1,WFS_NAME1,DIRECTION2,WFS_NAME2,STATE)
                                        =NEW_STATE ;

    WFS LEVEL FUNCTIONS :
BLOCK_AT(DIRECTION,WFS)=BLOCK ;
CONSTITUENT_AT(DIRECTION,WFS)=CONSTITUENT ;
DELETE(DIRECTION,WFS)=NEW_WFS ;
INSERT(DIRECTION,BLOCK,WFS)=NEW_WFS ;
CHANGE(DIRECTION,CONSTITUENT,WFS)=NEW_WFS ;
SKIP_BLOCK(DIRECTION,WFS)=NEW_WFS ;
INBLO(DIRECTION,WFS)=NEW_WFS ;
SHIFT(DIRECTION,WFS)=NEW_WFS ;
RESTORE(WFS)=NEW_WFS ;

SCANNER_ATTR_OF(WFS)=SCANNER_ATTRIBUTE ;
SET_SCANNER_ATTR(SCANNER_ATTRIBUTE,WFS)=NEW_WFS ;
```

```
    CONSTITUENT LEVEL FUNCTIONS :
TYPE_OF(CONSTITUENT)=TYPE ;
ATTR_OF(CONSTITUENT)=ATTRIBUTE ;
BITS_IN(CONSTITUENT)=BITSTRING ;
CHRS_IN(CONSTITUENT)=CHARACTERSTRING ;
NUMB_IN(CONSTITUENT)=NUMBER ;
NAME_IN(CONSTITUENT)=WFS_NAME ;

CONVERT_DATA(CONVERSION_MODE,CONSTITUENT)=NEW_CONSTITUENT ;
SET_ATTR(ATTRIBUTE,CONSTITUENT)=NEW_CONSTITUENT ;

ADDS(CONSTITUENT1,CONSTITUENT2)=NEW_CONSTITUENT ;
SUBS(CONSTITUENT1,CONSTITUENT2)=NEW_CONSTITUENT ;
MLTS(CONSTITUENT1,CONSTITUENT2)=NEW_CONSTITUENT ;
DIVS(CONSTITUENT1,CONSTITUENT2)=NEW_CONSTITUENT ;
ANDS(CONSTITUENT1,CONSTITUENT2)=NEW_CONSTITUENT ;
 ORS(CONSTITUENT1,CONSTITUENT2)=NEW_CONSTITUENT ;
NOTS(CONSTITUENT)=NEW_CONSTITUENT ;
CONCS_BITS(CONSTITUENT1,CONSTITUENT2)=NEW_CONSTITUENT ;
CONCS_CHRS(CONSTITUENT1,CONSTITUENT2)=NEW_CONSTITUENT ;
SPLIT_BITS1(CONSTITUENT)=NEW_CONSTITUENT ;
SPLIT_BITS2(CONSTITUENT)=NEW_CONSTITUENT ;
SPLIT_CHRS1(CONSTITUENT)=NEW_CONSTITUENT ;
SPLIT_CHRS2(CONSTITUENT)=NEW_CONSTITUENT ;

   SUBCONSTITUENT LEVEL FUNCTIONS :
OPPOSITE(DIRECTION)=NEW_DIRECTION ;
```

2.2. POSTULATES

FORMAL RULES AND CONVENTIONS :

' = ' IS A WEAK EQUALITY RELATION, I.E.
A=B IFF EITHER BOTH A AND B ARE DEFINED AND THE SAME,
OR BOTH A AND B ARE UNDEFINED ;

A=B & IS_SOMETHING(A) => IS_SOMETHING(B) ,
WHERE IS_SOMETHING IS ANY CATEGORY PREDICATE ;

POSTULATES :

```
IS_TYPE(TYPE) <=> TYPE=$B
                | TYPE=$C
                | TYPE=$D
                | TYPE=$P
                | TYPE=$R
                | TYPE=$LEFT_PARENT
                | TYPE=$RIGHT_PARENT ;

IS_DIRECTION(D) <=> D=LEFT | D=RIGHT ;
OPPOSITE(LEFT)=RIGHT ;
OPPOSITE(RIGHT)=LEFT ;
IS_DIRECTION(OPPOSITE(D)) <=> IS_DIRECTION(D) ;

IS_TYPE(TYPE_OF(C)) <=> IS_CONSTITUENT(C) ;
IS_BITS(BITS_IN(C)) <=> IS_CONSTITUENT(C) ;
IS_CHRS(CHRS_IN(C)) <=> IS_CONSTITUENT(C) ;
IS_NUMB(NUMB_OF(C)) <=> IS_CONSTITUENT(C) ;
IS_NAME(NAME_IN(C)) <=> IS_CONSTITUENT(C) ;

IS_CONSTITUENT(ADDS(C1,C2)) <=> TYPE_OF(C1)=$D
                                & TYPE_OF(C2)=$D
                               => TYPE_OF(ADDS(C1,C2))=$D ;
IS_CONSTITUENT(SUBS(C1,C2)) <=> TYPE_OF(C1)=$D
                                & TYPE_OF(C2)=$D
                               => TYPE_OF(SUBS(C1,C2))=$D ;
IS_CONSTITUENT(MLTS(C1,C2)) <=> TYPE_OF(C1)=$D
                                & TYPE_OF(C2)=$D
                               => TYPE_OF(MLTS(C1,C2))=$D ;
IS_CONSTITUENT(DIVS(C1,C2)) <=> TYPE_OF(C1)=$D
                                & TYPE_OF(C2)=$D
                                &¬NUMBER_IN(C2)=0
                               => TYPE_OF(DIVS(C1,C2))=$D ;
IS_CONSTITUENT(ANDS(C1,C2)) <=> TYPE_OF(C1)=$B
                                & TYPE_OF(C2)=$B
                               => TYPE_OF(ANDS(C1,C2))=$B ;
IS_CONSTITUENT( ORS(C1,C2)) <=> TYPE_OF(C1)=$B
                                & TYPE_OF(C2)=$B
                               => TYPE_OF( ORS(C1,C2))=$B ;
IS_CONSTITUENT(NOTS(C1))    <=> TYPE_OF(C1)=$B
                               => TYPE_OF(NOTS(C1))=$B ;
IS_CONSTITUENT(CONCS_BITS(C1,C2)) <=> TYPE(C1)=$B
```

```
                                        & TYPE(C2)=$B.
                    => TYPE_OF(CONCS_BITS(C1,C2))=$B ;
IS_CONSTITUENT(CONCS_CHRS(C1,C2)) <=> TYPE(C1)=$C
                                        & TYPE(C2)=$C
                    => TYPE_OF(CONCS_CHRS(C1,C2))=$C ;
IS_CONSTITUENT(SPLIT_BITS1(C)) <=> TYPE_OF(C)=$B
                    => TYPE_OF(SPLIT_BITS1(C))=$B ;
IS_CONSTITUENT(SPLIT_BITS2(C)) <=> TYPE_OF(C)=$B
                    => TYPE_OF(SPLIT_BITS2(C))=$B ;
IS_CONSTITUENT(SPLIT_CHRS1(C)) <=> TYPE_OF(C)=$C
                    => TYPE_OF(SPLIT_CHRS1(C))=$C ;
IS_CONSTITUENT(SPLIT_CHRS2(C)) <=> TYPE_OF(C)=$C
                    => TYPE_OF(SPLIT_CHRS2(C))=$C ;

IS_CONVERTIBLE(CONVERSION_MODE,CONSTITUENT)
    => IS_CONSTITUENT(CONVERT_DATA(CONVERSION_MODE,
                                   CONSTITUENT)) ;

IS_CONSTITUENT(CONSTITUENT) => IS_BLOCK(CONSTITUENT)
                    | TYPE_OF(CONSTITUENT)=$LEFT_PARENT
                    | TYPE_OF(CONSTITUENT)=$RIGHT_PARENT ;

IS_CONSTITUENT(CONSTITUENT_AT(DIRECTION,WFS))
    <=> IS_DIRECTION(DIRECTION) & IS_WFS(WFS) ;
IS_BLOCK(BLOCK_AT(DIRECTION,WFS))
    <=> IS_WFS(WFS) & DIRECTION=LEFT
      &¬TYPE_OF(CONSTITUENT_AT(DIRECTION,WFS))=$LEFT_PARENT
      | IS_WFS(WFS) & DIRECTION=RIGHT
      &¬TYPE_OF(CONSTITUENT_AT(DIRECTION,WFS))=
                                   $RIGHT_PARENT ;


IS_WFS(DELETE(DIRECTION,WFS))
    <=> IS_BLOCK(BLOCK_AT(DIRECTION,WFS)) ;
IS_WFS(INSERT(DIRECTION,BLOCK,WFS))
    <=> IS_DIRECTION(DIRECTION) & IS_BLOCK(BLOCK)
                                & IS_WFS(WFS) ;
IS_WFS(INSERT(DIRECTION,BLOCK,WFS))
     => BLOCK_AT(DIRECTION,INSERT(DIRECTION,BLOCK,WFS))
                                        =BLOCK ;
IS_WFS(INSERT(DIRECTION,BLOCK,WFS))
    => DELETE(DIRECTION,INSERT(DIRECTION,BLOCK,WFS))=WFS ;
IS_BLOCK(BLOCK_AT(DIRECTION,WFS))
     => INSERT(DIRECTION,BLOCK_AT(DIRECTION,WFS),
                         DELETE(DIRECTION,WFS))=WFS ;
IS_WFS(CHANGE(DIRECTION,CONSTITUENT,WFS))
    <=> IS_BLOCK(CONSTITUENT)
      & IS_BLOCK(CONSTITUENT_AT(DIRECTION,WFS))
      | IS_CONSTITUENT(CONSTITUENT)
      & TYPE_OF(CONSTITUENT)=TYPE_OF(CONSTITUENT_AT
                                   (DIRECTION,WFS)) ;
IS_BLOCK(CONSTITUENT)
      & IS_BLOCK(CONSTITUENT_AT(DIRECTION,WFS))
```

```
     => CHANGE(DIRECTION,CONSTITUENT,WFS)=INSERT(DIRECTION,
                    CONSTITUENT,DELETE(DIRECTION,WFS)) ;

SKIP_BLOCK(D,WFS)=
        INSERT(OPPOSITE(D),BLOCK_AT(D,WFS),DELETE(D,WFS)) ;

IS_WFS(INBLO(D,WFS)) <=> IS_DIRECTION(D) & IS_WFS(WFS) ;
IS_BLOCK(BLOCK_AT(D,WFS)
     => INBLO(D,WFS)=INBLO(D,SKIP_BLOCK(D,WFS)) ;
TYPE_OF(CONSTITUENT_AT(D,WFS))=$LEFT_PARENT & D=LEFT
    | TYPE_OF(CONSTITUENT_AT(D,WFS))=$RIGHT_PARENT & D=RIGHT
  => INBLO(D,WFS)=WFS ;

IS_WFS(SHIFT(DIRECTION,WFS)) <=> IS_DIRECTION(DIRECTION)
                                & IS_WFS(WFS) ;
IS_BLOCK(CONSTITUENT_AT(D,WFS))
     => SHIFT(D,WFS)=SKIP_BLOCK(D,WFS) ;
IS_WFS(SHIFT(D,WFS))
     => SHIFT(OPPOSITE(D),SHIFT(D,WFS))=WFS ;
CONSTITUENT_AT(OPPOSITE(D),SHIFT(D,WFS))=
                              CONSTITUENT_AT(D,WFS) ;
IS_WFS(INBLO(D1,WFS))
      & INBLO(D1,WFS)=INBLO(D1,SHIFT(D,WFS))
     => IS_BLOCK(CONSTITUENT_AT(D,WFS)) ;
SHIFT(D,INBLO(D,WFS))=SKIP_BLOCK(D,SHIFT(OPPOSITE(D),
                              INBLO(OPPOSITE(D),WFS))) ;

IS_WFS(RESTORE(WFS)) <=> IS_WFS(WFS) ;
RESTORE(RESTORE(WFS))=RESTORE(WFS) ;
IS_DIRECTION(D) & IS_WFS(WFS)
     => RESTORE(WFS)=RESTORE(SHIFT(D,WFS)) ;

IS_WFS(CONTENT(NAME,STATE)) => IS_NAME(NAME)
                              & IS_STATE(STATE) ;
IS_STATE(STATE) => IS_WFS(CONTENT(EXEC(STATE),STATE)) ;
CONTENT_OF_EXEC(STATE)=CONTENT(EXEC(STATE),STATE) ;
WFS_NAMED(NAME)=CONTENT(NAME,STATE)
                        WHERE  STATE  IS THE CURRENT STATE;

IS_STATE(KILL(NAME,STATE)) <=> IS_WFS(CONTENT(NAME,STATE)) ;
IS_STATE(CREATE(NAME,WFS,STATE))
    <=> IS_NAME(NAME) & IS_WFS(WFS) & IS_STATE(STATE)
      &¬IS_WFS(CONTENT(NAME,STATE)) ;
IS_STATE(CREATE(NAME,WFS,STATE))
     => CONTENT(NAME,CREATE(NAME,WFS,STATE))=WFS ;
IS_STATE(CREATE(NAME,WFS,STATE))
     => KILL(NAME,CREATE(NAME,WFS,STATE))=STATE ;
IS_WFS(CONTENT(NAME,STATE))
     => CREATE(NAME,CONTENT(NAME,STATE),KILL(NAME,STATE))
                                              =STATE ;

ALTER(NAME,WFS,STATE)=CREATE(NAME,WFS,KILL(NAME,STATE)) ;
```

```
MOVES(D1,NAME1,D2,NAME2,STATE)
    =ALTER(NAME1,DELETE(D1,CONTENT(NAME1,STATE)),

     ALTER(NAME2,INSERT(D2,
                BLOCK_AT(D1,CONTENT(NAME1,STATE)),
                CONTENT(NAME2,STATE)),
                 STATE)) ;
```

2.3. NUCLEOL IN PL/1

```
NUCLEOL:PROCEDURE OPTIONS(MAIN);
DECLARE(MOVE,COPY,RSTR,SHFT,CVRT,
        ADD,SUB,MLT,DIV,AND,OR,NOT,
        CONC,SPLT,TEST)
        ENTRY(BINARY FIXED) RETURNS(BINARY FIXED);
DECLARE $B                CHARACTER(1) INITIAL( 'B' ),
        $C                CHARACTER(1) INITIAL( 'C' ),
        $D                CHARACTER(1) INITIAL( 'D' ),
        $P                CHARACTER(1) INITIAL( 'P' ),
        $R                CHARACTER(1) INITIAL( 'R' ),
        $S                CHARACTER(1) INITIAL( 'S' ),
        $LEFT_PARENT      CHARACTER(1) INITIAL( '(' ),
        $RIGHT_PARENT     CHARACTER(1) INITIAL( ')' );
DECLARE XNONACTIVE CHARACTER(1) INITIAL( 'X' ),
        NONACTIVE  CHARACTER(1) INITIAL( 'N' ),
        SUCCESS    CHARACTER(1) INITIAL( 'S' ),
        FAILURE    CHARACTER(1) INITIAL( 'F' ),
         UNDEFIN    CHARACTER(1) INITIAL( 'U' ),
        DATA       CHARACTER(1) INITIAL( 'D' ),
        W          CHARACTER(1) INITIAL( 'W' );
DECLARE LEFT       CHARACTER(1) INITIAL( 'L' ),
        RIGHT      CHARACTER(1) INITIAL( 'R' ),
        NEUTRAL    CHARACTER(1) INITIAL( 'N' );
DECLARE STATE BINARY FIXED INITIAL(0);
DECLARE(NEW_STATE,STATE1) BINARY FIXED;
DECLARE ARG (3) BINARY FIXED,
        ARG1 BINARY FIXED DEFINED ARG(1),
        ARG2 BINARY FIXED DEFINED ARG(2),
        ARG3 BINARY FIXED DEFINED ARG(3);
DECLARE(ACTUAL_ARG1,ACTUAL_ARG2,ACTUAL_ARG3)
        BINARY FIXED;

DECLARE UNDEFINED BINARY FIXED INITIAL(-1);
DECLARE IS_STATE
    ENTRY(BINARY FIXED)
    RETURNS(BIT(1)),
        IS_WFS
    ENTRY(BINARY FIXED)
    RETURNS(BIT(1)),
        IS_BLOCK
    ENTRY(BINARY FIXED)
    RETURNS(BIT(1)),
        IS_CONSTITUENT
    ENTRY(BINARY FIXED)
    RETURNS(BIT(1)),
        IS_TYPE
    ENTRY(CHARACTER(1))
    RETURNS(BIT(1)),
        IS_DIRECTION
```

```
    ENTRY(CHARACTER(1))
    RETURNS(BIT(1)),
         IS_CONVERTIBLE
    ENTRY(BINARY FIXED,BINARY FIXED)
    RETURNS(BIT(1));
DECLARE CAN_PASS ENTRY(BIT(8),BIT(8)) RETURNS(BIT(1));

DECLARE NOCSTEP
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED);

DECLARE EVAL
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED);

DECLARE EXEC
    ENTRY(BINARY FIXED)
    RETURNS(CHARACTER(8)),
        CONTENT
    ENTRY(CHARACTER(8),BINARY FIXED)
    RETURNS(BINARY FIXED),
        WES_NAMED
    ENTRY(CHARACTER(8))
    RETURNS(BINARY FIXED),
        CONTENT_OF_EXEC
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE KILL
    ENTRY(CHARACTER(8),BINARY FIXED)
    RETURNS(BINARY FIXED),
        CREATE
    ENTRY(CHARACTER(8),BINARY FIXED,BINARY FIXED)
    RETURNS(BINARY FIXED),
        ALTER
    ENTRY(CHARACTER(8),BINARY FIXED,BINARY FIXED)
    RETURNS(BINARY FIXED),
        MOVES
    ENTRY(CHARACTER(1),CHARACTER(8),
          CHARACTER(1),CHARACTER(8),BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE PARSE_AND_SKIP
    ENTRY(BINARY FIXED,BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE EXECUTE_INSTRUCTION
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE  BLOCK_AT
    ENTRY(CHARACTER(1),BINARY FIXED)
    RETURNS(BINARY FIXED),
        CONSTITUENT_AT
    ENTRY(CHARACTER(1),BINARY FIXED)
    RETURNS(BINARY FIXED),
        DELETE
```

```
    ENTRY(CHARACTER(1),BINARY FIXED)
    RETURNS(BINARY FIXED),
        INSERT
    ENTRY(CHARACTER(1),BINARY FIXED,BINARY FIXED)
    RETURNS(BINARY FIXED),
        CHANGE
    ENTRY(CHARACTER(1),BINARY FIXED,BINARY FIXED)
    RETURNS(BINARY FIXED),
       (SHIFT,INBLD)
    ENTRY(CHARACTER(1),BINARY FIXED)
    RETURNS(BINARY FIXED),
        RESTORE
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE SCANNER_ATTR_OF
    ENTRY(BINARY FIXED)
    RETURNS(CHARACTER(1)),
        SET_SCANNER_ATTR
    ENTRY(CHARACTER(1),BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE TYPE_OF
    ENTRY(BINARY FIXED)
    RETURNS(CHARACTER(1)),
        ATTR_OF
    ENTRY(BINARY FIXED)
    RETURNS(CHARACTER(1)),
        BITS_IN
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED),
        CHRS_IN
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED),
        NUMB_IN
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED(31)),
        NAME_IN
    ENTRY(BINARY FIXED)
    RETURNS(CHARACTER(8));
DECLARE SET_ATTR
    ENTRY(CHARACTER(1),BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE CONVERT_DATA
    ENTRY(BINARY FIXED,BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE(ADDS,SUBS,MLTS,DIVS)
    ENTRY(BINARY FIXED,BINARY FIXED)
    RETURNS(BINARY FIXED);
DECLARE(ANDS,ORS)
    ENTRY(BINARY FIXED,BINARY FIXED)
    RETURNS(BINARY FIXED),
        NOTS
    ENTRY(BINARY FIXED)
    RETURNS(BINARY FIXED);
```

```
DECLARE(CONCS_BITS,CONCS_CHRS)
    ENTRY(BINARY FIXED,BINARY FIXED)
                RETURNS(BINARY FIXED);
DECLARE(SPLIT_BITS1,SPLIT_BITS2,SPLIT_CHRS1,SPLIT_CHRS2)
    ENTRY(BINARY FIXED) .
    RETURNS(BINARY FIXED);
DECLARE TESTS
    ENTRY(BINARY FIXED,BINARY FIXED,BINARY FIXED)
    RETURNS(BIT(1));
DECLARE NEW_CONSTITUENT BINARY FIXED;
DECLARE ERR_MSG
    ENTRY(CHARACTER(80));



     DO WHILE (IS_STATE(STATE));
        STATE=NUCSTEP(STATE);
     END;

     RETURN;




NUCSTEP:PROCEDURE(STATE)BINARY FIXED;

DECLARE STATE BINARY FIXED;
DECLARE NUS BINARY FIXED;

  NUS=CONSTITUENT_AT(RIGHT,CONTENT_OF_EXEC(STATE));
  IF TYPE_OF(NUS)=$C &
     ATTR_OF(NUS)='K'
  THEN DO; NEW_STATE=EXECUTE_INSTRUCTION(STATE);
           RETURN(NEW_STATE);
       END;
  IF TYPE_OF(NUS)=$LEFT_PARENT
  THEN DO;
         IF
         CAN_PASS(SCANNER_ATTR_OF(CONTENT_OF_EXEC(STATE)),
                                  ATTR_OF(NUS))
         THEN NEW_STATE=ALTER(EXEC(STATE),
                        SET_SCANNER_ATTR(NONACTIVE,SHIFT
                       (RIGHT,CONTENT_OF_EXEC(STATE))),
                                  STATE);
         ELSE NEW_STATE=MOVES(RIGHT,EXEC(STATE),
                                  LEFT,EXEC(STATE),
                                  STATE);


         RETURN(NEW_STATE);
       END;
  IF TYPE_OF(NUS)=$RIGHT_PARENT
```

```
  THEN DO;
         IF
         CAN_PASS(SCANNER_ATTR_OF(CONTENT_OF_EXEC(STATE)),
                                  ATTR_OF(NUS))
         THEN NEW=STATE=ALTER(EXEC(STATE),
                        SET_SCANNER_ATTR(NONACTIVE,SHIFT
                       (RIGHT,CONTENT_OF_EXEC(STATE))),
                                  STATE);
         ELSE NEW_STATE=ALTER(EXEC(STATE),
                                  INBLU(LEFT,CONTENT_OF_EXEC(
                                                  STATE)),
                                  STATE);
         RETURN(NEW_STATE);
       END;
  IF TYPE_OF(NUS)=$R &
     ATTR_OF(NUS)=NEUTRAL
  THEN DO;
         NEW_STATE=ALTER(NAME_IN(NUS),
                        SET_SCANNER_ATTR(NONACTIVE,CONTENT(
                  NAME_IN(NUS),STATE)),
                         ALTER(EXEC(STATE),
                        SET_SCANNER_ATTR(DATA,
                  CONTENT_OF_EXEC(ALTER(EXEC(STATE),
                                        SHIFT(RIGHT,
                  CONTENT_OF_EXEC(STATE)),
                                        STATE))),
                                   STATE));
         RETURN(NEW_STATE);
       END;
  /*MARKER CASE*/
  NEW_STATE=ALTER(EXEC(STATE),
                  SHIFT(RIGHT,CONTENT_OF_EXEC(STATE)),
                  STATE);
  RETURN(NEW_STATE);

END NUCSTEP;



EXECUTE_INSTRUCTION:PROCEDURE(STATE) BINARY FIXED;
DECLARE STATE BINARY FIXED;
  DECLARE KEY CHARACTER(4);
  KEY=CHPS_IN(CONSTITUENT_AT(RIGHT,CONTENT_OF_EXEC(STATE)));
  IF KEY='MOVE' THEN RETURN(MOVE(PARSE_AND_SKIP(2,STATE)));
  IF KEY='COPY' THEN RETURN(COPY(PARSE_AND_SKIP(2,STATE)));
  IF KEY='SHFT' THEN RETURN(SHFT(PARSE_AND_SKIP(1,STATE)));
  IF KEY='RSTR' THEN RETURN(RSTR(PARSE_AND_SKIP(1,STATE)));
  IF KEY='TEST' THEN RETURN(TEST(PARSE_AND_SKIP(3,STATE)));
  IF KEY='ADD ' THEN RETURN(ADD (PARSE_AND_SKIP(3,STATE)));
  IF KEY='SUB ' THEN RETURN( SUB(PARSE_AND_SKIP(3,STATE)));
  IF KEY='MLT ' THEN RETURN( MLT(PARSE_AND_SKIP(3,STATE)));
  IF KEY='DIV ' THEN RETURN( DIV(PARSE_AND_SKIP(3,STATE)));
```

```
   IF KEY='AND ' THEN RETURN( AND(PARSE_AND_SKIP(3,STATE)));
   IF KEY='OR  ' THEN RETURN(  OR(PARSE_AND_SKIP(3,STATE)));
   IF KEY='NOT ' THEN RETURN( NOT(PARSE_AND_SKIP(2,STATE)));
   IF KEY='CONC' THEN RETURN(CONC(PARSE_AND_SKIP(3,STATE)));
   IF KEY='SPLT' THEN RETURN(SPLT(PARSE_AND_SKIP(2,STATE)));
   IF KEY='CVRT' THEN RETURN(CVRT(PARSE_AND_SKIP(1,STATE)));

   CALL ERR_MSG('KEY WORD IS NOT PROPER');
   RETURN(UNDEFINED);
END EXECUTE_INSTRUCTION;




PARSE_AND_SKIP:PROCEDURE(NUMBER,STATE) BINARY FIXED;
DECLARE NUMBER BINARY FIXED,
        STATE BINARY FIXED;
   DECLARE STATE1 BINARY FIXED;
   STATE1=ALTER(EXEC(STATE),
                SHIFT(RIGHT,CONTENT_OF_EXEC(STATE)),
                STATE);
   DO N=1 TO NUMBER;
      ARG(N)=BLOCK_AT(RIGHT,CONTENT_OF_EXEC(STATE1));
      IF ¬IS_DEFINED(ARG(N))
      THEN DO;
        CALL ERR_MSG('NOT ENOUGH ARGUMENTS IN AN INSTRUCTION'
                                                          );
        RETURN(UNDEFINED);
           END;
      STATE1=MOVES(RIGHT,EXEC(STATE),
                   LEFT,EXEC(STATE),
                   STATE);



   END;
   RETURN(STATE1);
END PARSE_AND_SKIP;




MOVE:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
```

```
      IF TYPE_OF(ARG1)¬=$R
       | TYPE_OF(ARG2)¬=$R
      THEN CALL ERR_MSG(
           'TYPE OF ARG1 OR ARG2 IN MOVE IS NO GOOD');
           RETURN(UNDEFINED);
      IF NAME_IN(ARG2)='SYS.FSL'
      THEN DO;
           IF IS_DIRECTION(ATTR_OF(ARG1))
           THEN RETURN(ALTER(NAME_IN(ARG1),
                             DELETE(ATTR_OF(ARG1),WFS_NAMED
                    (NAME_IN(ARG1))),
                             STATE));

           RETURN(KILL(NAME_IN(ARG1),STATE));
           END;
      IF IS_DIRECTION(ATTR_OF(ARG1))
       & IS_DIRECTION(ATTR_OF(ARG2))
      THEN RETURN(
                  MOVES(ATTR_OF(ARG1),NAME_IN(ARG1),
                                ATTR_OF(ARG2),NAME_IN(ARG2),
                                STATE)
                  );



      IF ¬IS_DIRECTION(ATTR_OF(ARG1))
       & ¬IS_DIRECTION(ATTR_OF(ARG2))
      THEN RETURN(CREATE(NAME_IN(ARG2),
                         WFS_NAMED(NAME_IN(ARG1)),
                         KILL(NAME_IN(ARG1),STATE)));
      CALL ERR_MSG('ARG1 OR ARG2 IN MOVE IS NO GOOD');
      RETURN(UNDEFINED);
END MOVE;



COPY:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
      IF TYPE_OF(ARG2)¬=$R
      THEN RETURN(UNDEFINED);
      IF TYPE_OF(ARG1)=$R
      THEN DO;
           IF IS_DIRECTION(ATTR_OF(ARG1))
            & IS_DIRECTION(ATTR_OF(ARG2))
           THEN RETURN(ALTER(NAME_IN(ARG2),
                             INSERT(ATTR_OF(ARG2),BLOCK_AT(
                    ATTR_OF(ARG1),WFS_NAMED(NAME_IN(ARG1))),
                    WFS_NAMED(NAME_IN(ARG2))),
                             STATE));
           IF ¬IS_DIRECTION(ATTR_OF(ARG1))
            & ¬IS_DIRECTION(ATTR_OF(ARG2))
```

```
          THEN DO;
           IF ¬IS_WFS(CONTENT(NAME_IN(ARG2),STATE))
               THEN RETURN(
        CREATE(NAME_IN(ARG2),WFS_NAMED(NAME_IN(ARG1)),STATE)
                          );
               CALL ERR_MSG(
        'ATTEMPT TO CREATE SECOND WFS WITH THE SAME NAME'
                          );
               RETURN(UNDEFINED);
               END;
         CALL ERR_MSG('ARG1 OR ARG2 IN COPY IS NO GOOD');
          RETURN(UNDEFINED);
          END;
     RETURN(ALTER(NAME_IN(ARG2),
                  INSERT(ATTR_OF(ARG2),ARG1,WFS_NAMED(
                                NAME_IN(ARG2))),
                  STATE));
END COPY;




RSTR:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
     IF TYPE_OF(ARG1)¬=PR
     THEN DO;

          CALL ERR_MSG('ARG1 IN RSTR IS NOT A REFERENCE');
          RETURN(UNDEFINED);
          END;
     IF IS_DIRECTION(ATTR_OF(ARG1))
     THEN DO;

           IF NAME_IN(ARG1)=EXEC(STATE)
          THEN STATE1=ALTER(EXEC(STATE),
                       SET_SCANNER_ATTR(NONACTIVE,
                                CONTENT_OF_EXEC(STATE)),
                            STATE);
          ELSE STATE1=STATE;
          RETURN(ALTER(NAME_IN(ARG1),
                       INBLD(ATTR_OF(ARG1),CONTENT(NAME_IN
                                (ARG1),STATE1)),
                       STATE1));
          END;
     IF NAME_IN(ARG1)=EXEC(STATE)
     THEN STATE1=ALTER(EXEC(STATE),
                       SET_SCANNER_ATTR(NONACTIVE,
                                CONTENT_OF_EXEC(STATE)),
                       STATE);
     ELSE STATE1=STATE;
     RETURN(ALTER(NAME_IN(ARG1),
```

```
                    RESTORE(CONTENT(NAME_IN(ARG1),STATE1)),
                    STATE1));
END RSTR;



SHFT:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
  IF TYPE_OF(ARG1)=$R &
     IS_DIRECTION(ATTR_OF(ARG1)) &
      NAME_IN(ARG1)¬=EXEC(STATE)
  THEN RETURN(
              ALTER(NAME_IN(ARG1),
                    SHIFT(ATTR_OF(ARG1),WFS_NAMED(NAME_IN
                                                  (ARG1))),
                    STATE)
             );
  ELSE CALL ERR_MSG('ARG1 IN SHFT IS NO GOOD');
       RETURN(UNDEFINED);
END SHFT;



CVRT:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
DECLARE TYPE_ATTRIBUTE_TABLE(28) CHARACTER(2) INITIAL
              ( 'B ' , 'BS' ,
                'C ' , 'CR' , 'CM' , 'CS' ,
                'D ' , 'DS' ,
                'P ' ,'PB' , 'PC' , 'PD' , 'PR' ,
                'R ' , 'RL' , 'RR' ,
          '( ' , '(N' , '(S' , '(F' , '(U' , '(D' ,
          ') ' , ')N' , ')S' , ')F' , ')U' , ')D' );

DECLARE ACTUAL_CONVERSION_MODE_TABLE(10) CHARACTER(2)
       INITIAL( 'BC' , 'BD' ,
                'CB' , 'CD' , 'CP' , 'CR' ,
                'DB' , 'DC' ,
                'PC' ,
                'RC' );

  DECLARE  CONVERSION_MODE CHARACTER(3),
          (TYPE_CHR,ATTR_CHR,TYPE_ORG,ATTR_ORG)CHARACTER(1),
          (TYPE_ATTR,TYPE_TYPE) CHARACTER(2);
  IF TYPE_OF(ARG2)=$R &
```

```
    IS_DIRECTION(ATTR_OF(ARG2))
THEN ACTUAL_ARG2=CONSTITUENT_AT(ATTR_OF(ARG1),
                                WFS_NAMED(NAME_IN(ARG2)));
ELSE DO; CALL ERR_MSG('ARG2 IN CVRT IS NO GOOD');
         RETURN(UNDEFINED);
     END;

  ACTUAL_ARG1=EVAL(ARG1);
IF TYPE_OF(ARG1)=$C
THEN CONVERSION_MODE=CHRS_IN(ARG1) ;
ELSE DO;
  IF TYPE_OF(ARG1)=$R &
     IS_DIRECTION(ATTR_OF(ARG1))
  THEN ACTUAL_ARG1=CONSTITUENT_AT(ATTR_OF(ARG1),
                                  WFS_NAMED(NAME_IN(ARG1)));
  ELSE DO; CALL ERR_MSG('ARG1 IN CVRT IS NO GOOD');
           RETURN(UNDEFINED);
       END;

  IF TYPE_OF(ACTUAL_ARG1)=$C
  THEN CONVERSION_MODE=CHRS_IN(ACTUAL_ARG1);
ELSE DO;CALL ERR_MSG(
       'ARG1 OR ACTUAL_ARG1 IN CVRT IS NO GOOD');
         RETURN(UNDEFINED);
       END;
     END;

TYPE_CHR=SUBSTR(CONVERSION_MODE,1,1);
ATTR_CHR=SUBSTR(CONVERSION_MODE,3,1);
TYPE_ORG=TYPE_OF(ACTUAL_ARG2);
ATTR_ORG=ATTR_OF(ACTUAL_ARG2);
IF TYPE_CHR=' ' THEN TYPE_CHR=TYPE_ORG;
IF ATTR_CHR=' ' THEN ATTR_CHR=ATTR_ORG;
TYPE_ATTR=TYPE_CHR||ATTR_CHR;
TYPE_TYPE=TYPE_ORG||TYPE_CHR;

CHECK_CONVERSION_MODE:
DO I=1 TO DIM(TYPE_ATTRIBUTE_TABLE,1);
   IF TYPE_ATTR=TYPE_ATTRIBUTE_TABLE(I)
   THEN GOTO CVRT1;
END CHECK_CONVERSION_MODE;
CALL ERR_MSG('ILLEGAL CONVERSION MODE IN CVRT');
RETURN(UNDEFINED);

CVRT1:
IF TYPE_CHR¬=TYPE_ORG
THEN DO;
FIND_ACTUAL_CONVERSION_MODE:
DO I=1 TO DIM(ACTUAL_CONVERSION_MODE_TABLE,1);
   IF TYPE_TYPE=ACTUAL_CONVERSION_MODE_TABLE(I)
   THEN GO TO CVRT2;
END FIND_ACTUAL_CONVERSION_MODE;
CALL ERR_MSG('ILLEGAL CONVERSION REQUESTED IN CVRT');
```

```
   RETURN(UNDEFINED);

   CVRT2:
   IF IS_CONVERTIBLE(I,ACTUAL_ARG2)
   THEN
   ACTUAL_ARG2=CONVERT_DATA(I,ACTUAL_ARG2);
   ELSE DO;  CALL ERR_MSG(
   'ACTUAL_ARG2 IN CONVERT IS NOT CONVERTIBLE'    );
               RETURN(UNDEFINED);
        END;
        END;
   IF ATTR_CHR¬=ATTR_ORG
   THEN ACTUAL_ARG2=SET_ATTR(ATTR_CHR,ACTUAL_ARG2);
   RETURN(
          ALTER(NAME_IN(ARG2),
                        CHANGE(ATTR_OF(ARG2),ACTUAL_ARG2,
                              WFS_NAMED(NAME_IN(ARG1))),
                        STATE));
END CVRT;




ADD:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
     ACTUAL_ARG1=EVAL(ARG1);
     ACTUAL_ARG2=EVAL(ARG2);
     IF TYPE_OF(ACTUAL_ARG1)=$D
      & TYPE_OF(ACTUAL_ARG2)=$D
      & TYPE_OF(ARG3)=$R
      & IS_DIRECTION(ATTR_OF(ARG3))
     THEN DO;
          NEW_CONSTITUENT=ADDS(ACTUAL_ARG1,ACTUAL_ARG2);


          RETURN(ALTER(NAME_IN(ARG3),
                        INSERT(ATTR_OF(ARG3),NEW_CONSTITUENT,
                                    WFS_NAMED(NAME_IN(ARG3))),
                        STATE));
          END;
     CALL ERR_MSG(
         'AN ARG OR AN ACTUAL_ARG IN ADD IS NO GOOD');
     RETURN(UNDEFINED);
END ADD;



SUB:PROCEDURE(STATE)BINARY FIXED;
```

```
DECLARE STATE BINARY FIXED;
     ACTUAL_ARG1=EVAL(ARG1);
     ACTUAL_ARG2=EVAL(ARG2);
     IF TYPE_OF(ACTUAL_ARG1)=$D
      & TYPE_OF(ACTUAL_ARG2)=$D
      & TYPE_OF(ARG3)=$R
      & IS_DIRECTION(ATTR_OF(ARG3))
     THEN DO;
          NEW_CONSTITUENT=SUBS(ACTUAL_ARG1,ACTUAL_ARG2);


          RETURN(ALTER(NAME_IN(ARG3),
                       INSERT(ATTR_OF(ARG3),NEW_CONSTITUENT,
                              WFS_NAMED(NAME_IN(ARG3))),
                       STATE));
          END;
     CALL ERR_MSG(
         'AN ARG OR AN ACTUAL_ARG IN SUB IS NO GOOD');
     RETURN(UNDEFINED);
END SUB;



MLT:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
     ACTUAL_ARG1=EVAL(ARG1);
     ACTUAL_ARG2=EVAL(ARG2);
     IF TYPE_OF(ACTUAL_ARG1)=$D
      & TYPE_OF(ACTUAL_ARG2)=$D
      & TYPE_OF(ARG3)=$R
      & IS_DIRECTION(ATTR_OF(ARG3))
     THEN DO;
          NEW_CONSTITUENT=MLTS(ACTUAL_ARG1,ACTUAL_ARG2);


          RETURN(ALTER(NAME_IN(ARG3),
                       INSERT(ATTR_OF(ARG3),NEW_CONSTITUENT,
                              WFS_NAMED(NAME_IN(ARG3))),
                       STATE));
          END;
     CALL ERR_MSG(
         'AN ARG OR AN ACTUAL_ARG IN MLT IS NO GOOD');
     RETURN(UNDEFINED);
END MLT;



DIV:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
     ACTUAL_ARG1=EVAL(ARG1);
     ACTUAL_ARG2=EVAL(ARG2);
     IF TYPE_OF(ACTUAL_ARG1)=$D
      & TYPE_OF(ACTUAL_ARG2)=$D
```

```
      & TYPE_OF(ARG3)=$R
      & IS_DIRECTION(ATTR_OF(ARG3))
     THEN DO;
          IF NUMB_IN(ACTUAL_ARG2)¬=0
          THEN DO;
          NEW_CONSTITUENT=DIVS(ACTUAL_ARG1,ACTUAL_ARG2);


          RETURN(ALTER(NAME_IN(ARG3),
                       INSERT(ATTR_OF(ARG3),NEW_CONSTITUENT,
                                 WFS_NAMED(NAME_IN(ARG3))),
                       STATE));
               END;
          CALL ERR_MSG('ACTUAL_ARG2 IN DIV IS 0');
          RETURN(UNDEFINED);
          END;
     CALL ERR_MSG(
         'AN ARG OR AN ACTUAL_ARG IN DIV IS NO GOOD');
     RETURN(UNDEFINED);
END DIV;



AND:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
     ACTUAL_ARG1=EVAL(ARG1);
     ACTUAL_ARG2=EVAL(ARG2);
     IF TYPE_OF(ACTUAL_ARG1)=$B
      & TYPE_OF(ACTUAL_ARG2)=$B
      & TYPE_OF(ARG3)=$R
      & IS_DIRECTION(ATTR_OF(ARG3))
     THEN DO;
          NEW_CONSTITUENT=ANDS(ACTUAL_ARG1,ACTUAL_ARG2);


          RETURN(ALTER(NAME_IN(ARG3),
                       INSERT(ATTR_OF(ARG3),NEW_CONSTITUENT,
                                 WFS_NAMED(NAME_IN(ARG3))),
                       STATE));
          END;
     CALL ERR_MSG(
         'AN ARG OR AN ACTUAL_ARG IN AND IS NO GOOD');
     RETURN(UNDEFINED);
END AND;



OR :PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
     ACTUAL_ARG1=EVAL(ARG1);
     ACTUAL_ARG2=EVAL(ARG2);
     IF TYPE_OF(ACTUAL_ARG1)=$B
      & TYPE_OF(ACTUAL_ARG2)=$B
```

```
      & TYPE_OF(ARG3)=$R
      & IS_DIRECTION(ATTR_OF(ARG3))
     THEN DO;
          NEW_CONSTITUENT= ORS(ACTUAL_ARG1,ACTUAL_ARG2);


          RETURN(ALTER(NAME_IN(ARG3),
                       INSERT(ATTR_OF(ARG3),NEW_CONSTITUENT,
                                  WFS_NAMED(NAME_IN(ARG3))),
                       STATE));
          END;
     CALL ERR_MSG(
         'AN ARG OR AN ACTUAL_ARG IN OR  IS NO GOOD');
     RETURN(UNDEFINED);
END OR ;



NOT:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
     ACTUAL_ARG1=EVAL(ARG1);
     IF TYPE_OF(ACTUAL_ARG1)=$B
      & TYPE_OF(ARG2)=$R
      & IS_DIRECTION(ATTR_OF(ARG2))
     THEN DO;
          NEW_CONSTITUENT=NOTS(ACTUAL_ARG1);

          RETURN(ALTER(NAME_IN(ARG3),
                       INSERT(ATTR_OF(ARG2),NEW_CONSTITUENT,
                                  WFS_NAMED(NAME_IN(ARG2))),
                       STATE));
          END;
     CALL ERR_MSG(
         'AN ARG OR AN ACTUAL_ARG IN NOT IS NO GOOD');
     RETURN(UNDEFINED);
END NOT;



CONC:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
      IF TYPE_OF(ARG3)¬=$R
       | ¬IS_DIRECTION(ATTR_OF(ARG3))
      THEN DO; CALL ERR_MSG('ARG3 IN CONC IS NO GOOD');
               RETURN(UNDEFINED);
           END;
      ACTUAL_ARG1=EVAL(ARG1);
      ACTUAL_ARG2=EVAL(ARG2);
      IF TYPE_OF(ACTUAL_ARG1)=$B
       & TYPE_OF(ACTUAL_ARG2)=$B
      THEN
```

```
          NEW_CONSTITUENT=CONCS_BITS(ACTUAL_ARG1,
                                     ACTUAL_ARG2);

      ELSE DO;
      IF TYPE_OF(ACTUAL_ARG1)=$C
       & TYPE_OF(ACTUAL_ARG2)=$C
      THEN
          NEW_CONSTITUENT=CONCS_CHRS(ACTUAL_ARG1,
                                     ACTUAL_ARG2);

      ELSE DO;CALL ERR_MSG(
           'ACTUAL_ARG1 OR ACTUAL_ARG2 IN CONC IS NO GOOD');
              RETURN(UNDEFINED);
           END;
           END;
      RETURN(ALTER(NAME_IN(ARG3),
                   INSERT(ATTR_OF(ARG3),NEW_CONSTITUENT,
                          WFS_NAMED(NAME_IN(ARG3))),
                   STATE));
END CONC;


SPLT:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
      IF TYPE_OF(ARG1)=$R
       & IS_DIRECTION(ATTR_OF(ARG1))
       & TYPE_OF(ARG2)=$R
       & IS_DIRECTION(ATTR_OF(ARG2))
      THEN DO;
           ACTUAL_ARG1=CONSTITUENT_AT(ATTR_OF(ARG1),
                       WFS_NAMED(NAME_IN(ARG1)));
           IF TYPE_OF(ACTUAL_ARG1)=$B
           THEN RETURN(ALTER(NAME_IN(ARG2),
                             INSERT(ATTR_OF(ARG2),
                             SPLIT_BITS2(ACTUAL_ARG1),

                              WFS_NAMED(NAME_IN(ARG2))),
                             ALTER(NAME_IN(ARG1),
                                   CHANGE(ATTR_OF(ARG1),
                             SPLIT_BITS1(ACTUAL_ARG1),

                               WFS_NAMED(NAME_IN(ARG1))),
                                   STATE)));
           IF TYPE_OF(ACTUAL_ARG1)=$C
           THEN RETURN(ALTER(NAME_IN(ARG2),
                             INSERT(ATTR_OF(ARG2),
                             SPLIT_CHRS2(ACTUAL_ARG1),

                              WFS_NAMED(NAME_IN(ARG2))),
                             ALTER(NAME_IN(ARG1),
                                   CHANGE(ATTR_OF(ARG1),
                             SPLIT_CHRS1(ACTUAL_ARG1),
```

```
                                    WFS_NAMED(NAME_IN(ARG1))),

                                        STATE)));
            CALL ERR_MSG('ACTUAL_ARG1 IN SPLT IS NO GOOD');
            RETURN(UNDEFINED);
            END;
      CALL ERR_MSG('ARG1 OR ARG2 IN SPLT IS NO GOOD');
      RETURN(UNDEFINED);
END SPLT;



TEST:PROCEDURE(STATE)BINARY FIXED;
DECLARE STATE BINARY FIXED;
DECLARE COMPARISON_MODE_TABLE(22) CHARACTER(4) INITIAL
                ( '  = ' , ' ¬= ' , ' <  ' , ' <= ' ,
                  ' >= ' , '  > ' ,
                  '  =A' , ' ¬=A' , '  =T' , ' ¬=T' ,
                  'A = ' , 'A¬= ' , 'T = ' , 'T¬= ' ,
                  'A =A' , 'A¬=A' , 'A =T' , 'A¬=T' ,
                  'T =A' , 'T¬=A' , 'T =T' , 'T¬=T' );
      DECLARE COMPARISON_MODE CHARACTER(4);
      DECLARE TST BIT(1);
      ACTUAL_ARG1=EVAL(ARG1);
      ACTUAL_ARG2=EVAL(ARG2);
      ACTUAL_ARG3=EVAL(ARG3);
      IF TYPE_OF(ACTUAL_ARG2)=$C
      THEN CHECK_COMPARISON_MODE:DO;
            COMPARISON_MODE=CHARS_IN(ACTUAL_ARG2);
            DO I=1 TO DIM(COMPARISON_MODE_TABLE,1);
            IF COMPARISON_MODE=COMPARISON_MODE_TABLE(I)
            THEN GOTO TEST1;
            END CHECK_COMPARISON_MODE;
      CALL ERR_MSG('COMPARISON_MODE IN TEST IS NO GOOD');
      RETURN(UNDEFINED);
      TEST1:
     IF I<=6 THEN DO;
      IF TYPE_OF(ACTUAL_ARG1)=TYPE_OF(ACTUAL_ARG2)
      THEN DO;
            TST=TESTS(I,ACTUAL_ARG1,ACTUAL_ARG3);


            GOTO TEST2;
            END;
      TEST_UNDEFINED:
            RETURN(ALTER(EXEC(STATE),
                        SET_SCANNER_ATTR(UNDEFIN,
                                    CONTENT_OF_EXEC(STATE)),
                        STATE));
                  END;
      IF I<=10 THEN
```

```
      DO;IF TYPE_OF(ACTUAL_ARG1)=$C
         THEN GOTO TEST3;
         ELSE GO TO TEST_UNDEFINED;
      END;
      IF I<=14
      THEN DO;IF TYPE_OF(ACTUAL_ARG3)=$C
              THEN GOTO TEST3;
              ELSE GOTO TEST_UNDEFINED;
           END;
      TEST3:
           TET=TESTS(I,ACTUAL_ARG1,ACTUAL_ARG3);
      TEST2:
      IF TET
      THEN RETURN(ALTER(EXEC(STATE),
                        SET_SCANNER_ATTR(SUCCESS,
                                         CONTENT_OF_EXEC(STATE)),
                        STATE));
           RETURN(ALTER(EXEC(STATE),
                        SET_SCANNER_ATTR(FAILURE,
                                         CONTENT_OF_EXEC(STATE)),
                        STATE));
END TEST;



EVAL:PROCEDURE(ARG)BINARY FIXED;
DECLARE ARG BINARY FIXED;
      IF TYPE_OF(ARG)=$R
       & IS_DIRECTION(ATTR_OF(ARG))
      THEN RETURN(CONSTITUENT_AT(ATTR_OF(ARG),
                                 WFS_NAMED(NAME_IN(ARG))));
      RETURN(ARG);
END EVAL;



END NUCLEUS;
```

3. NUCLEUL PROGRAMMING LANGUAGE AS A MODEL FOR NUCLEUL

NOTATION:

<A> DENOTES A SUBSET OF THE SET OF ALL STRINGS OVER ALL KEYPUNCH SYMBOLS ;
<A*> ::= <A> | <A><A*> ;
<A*?> ::= <A*> | <EMPTY> ;
<A,I> DENOTES AN ELEMENT OF <A>, I MAY BE SUBSTITUTED BY A POSITIVE INTEGER ;
<A*,I> DENOTES AN ELEMENT OF <A*> ;
<A*?,I> DENOTES AN ELEMENT OF <A*?> ;

DEFINE JAM TO BE THE UNION OF THE SET OF ALL STRINGS OVER ALL KEYPUNCH SYMBOLS AND FINITE COLLECTIONS OF SUCH STRINGS ;

3.1. NUCLEOL PROGRAMMING LANGUAGE SYNTAX

```
<WFS> ::= <BLOCK WITH SCANNER> | <$S><BLOCK> | <$S>
<BLOCK WITH SCANNER> ::= <$(><BLOCK*?><$S><BLOCK*?><$)> |
    <$(><BLOCK*?><BLOCK WITH SCANNER><BLOCK*?><$)>
<BLOCK> ::= <ORDINARY CONSTITUENT> | <$(><BLOCK*?><$)>
<CONSTITUENT> ::= <ORDINARY CONSTITUENT> |
                  <$(> | <$)> | <$S>
<ORDINARY CONSTITUENT> ::= <$B> | <$C> | <$D> | <$P> | <$R>
<$B> ::= $B<BA>'<BITSTRING>'
<$C> ::= $C<CA>'<CHARACTERSTRING>'
<$D> ::= $D<DA>'<NUMBER>'
<$P> ::= $P<TA>'<CHARACTERSTRING>'
<$R> ::= $R<RA>'<WFS NAME>'

<$(> ::= $(<PA>
<$)> ::= $)<PA>
<$S> ::= $S<SA>'<WFS NAME>'

<BA> ::= <BLANK> | S
<CA> ::= <BLANK> | K | M | S
<DA> ::= <BLANK> | S
<TA> ::= <BLANK> | B | C | D | R
<RA> ::= <BLANK> | L | R
<PA> ::= <BLANK> | D | F | N | S | U | X
                AND COMBINATIONS OF  F, N, S, AND U
<SA> ::= D | F | N | S | U

<BITSTRING> ::= <BIT*?>
<BIT> ::= 0 | 1
<CHARACTERSTRING> ::= <CHARACTER*?>
<CHARACTER> ::= <BLANK> | <REST> | <LETTER> | <DIGIT>
<BLANK> ::= A SINGLE SPACE
<REST> ::= . | < | ( | + | | | & | $ | * | ) | ; | ¬ | -
         | / | , | % | _ | > | ? | : | # | @ | ' | = | "
<LETTER> ::= A | B | C | D | E | F | G | H | I | J | K | L
           | M | N | O | P | Q | R | S | T | U | V | W | X
           | Y | Z
<DIGIT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<NUMBER> ::= <DIGIT*> | +<DIGIT*> | -<DIGIT*>
<WFS NAME> ::= SEQUENCE OF LETTERS, DIGITS, AND _
               BEGINNING WITH LETTER AND NOT LONGER
               THEN 8 CHARACTERS.

<INSTRUCTION> ::= <MOVE> | <COPY> | <SHFT> | <RSTR> |
                  <CVRT> | <ADD> | <SUB> | <MLT> | <DIV> |
                  <AND> | <OR> | <NOT> | <CONC> | <SPLT> |
                  <TEST>

<MOVE> ::= $CK'MOVE' ( <SR><SR> | <WR><WR> )
```

```
<COPY> ::= $CK'COPY' ( <BLOCK><SR> |
                       <SR> <WR> |
                       <WR> <WR> )
<SHFT> ::= $CK'SHFT' <SR>
<RSTR>  ::= $CK'RSTR' ( <SR> | <WR> )
<CVRT> ::= $CK'CVRT' ( <T/A> | <SR> ) <SR>
<ADD> ::= $CK'ADD' ( <$D> | <SR> ) ( <$D> | <SR> ) <SR>
<SUB> ::= $CK'SUB' ( <$D> | <SR> ) ( <$D> | <SR> ) <SR>
<MLT> ::= $CK'MLT' ( <$D> | <SR> ) ( <$D> | <SR> ) <SR>
<DIV> ::= $CK'DIV' ( <$D> | <SR> ) ( <$D> | <SR> ) <SR>
<AND> ::= $CK'AND' ( <$B> | <SR> ) ( <$B> | <SR> ) <SR>
<OR>  ::= $CK'OR'  ( <$B> | <SR> ) ( <$B> | <SR> ) <SR>
<NOT> ::= $CK'NOT' ( <$B> | <SR> ) <SR>
<CONC> ::= $CK'CONC' ( <$B> | <SR> ) ( <$B> | <SR> ) <SR>
         | $CK'CONC' ( <$C> | <SR> ) ( <$C> | <SR> ) <SR>
<SPLT> ::= $CK'SPLT' <SR> <SR>
<TEST> :: $CK'TEST' ( <$B> | <$C> | <$D> | <$P> | <SR> )
            ( <TEST MODE> | <SR> )
            ( <$B> | <$C> | <$D> | <$P> | <SR> )

<SR> ::= $RL'<NFS NAME>'  |  $RR'<NFS NAME>'
<WR> ::= $R '<NFS NAME>'

<TYPE> ::= B | C | D | P | R | S | ( | )
<T/A> ::= $C'B/<BA>' | $C'C/<CA>' | $C'D/<DA>' | $C'P/<TA>'
        | $C'R/<RA>' | $C'(/<PA>' | $C')/<PA>'
        | $C' /<BA>' | $C' /<CA>' | $C' /<DA>' | $C' /<TA>'
        | $C' /<RA>' | $C' /<PA>'
<TEST MODE> ::= $C'  = '  |  $C' ¬= '  |  $C' <  '  |
                $C' <= '  |  $C' >  '  |  $C' >= '  |
                $C'  =A'  |  $C' ¬=A'  |
                $C'  =T'  |  $C' ¬=T'  |
                $C'A =A'  |  $C'A¬=A'  |
                $C'A =T'  |  $C'A¬=T'  |
                $C'T =T'  |  $C'T¬=T'  |
                $C'T =A'  |  $C'T¬=A'
```

3.2. MAPPING OF NUCLEOL INTO NUCLEOL PROGRAMMING LANGUAGE

TO DEFINE AN INTERPRETATION OF NUCLEOL FORMAL SYSTEM
<ALPHABET,TERMS,FORMULAS> IN NUCLEOL PROGRAMMING LANGUAGE
I FIRST DEFINE A MAP @ ON ALPHABET AS FOLLOWS:

@ MAPS VARIABLES INTO THE SET OF VARIABLES RANGING OVER JAM;
/* JAM WAS DEFINED IN 3. */

@ MAPS BASIC PREDICATES INTO THE SET OF SUBSETS OF JAM
UNION THE CARTESIAN PRODUCT JAM X JAM
WHERE SOME OF THEM ARE DEFINED AS FOLLOWS:

@IS_STATE IS DEFINED TO BE THE SET OF ALL FINITE COLLECTIONS
OF ELEMENTS OF <WFS> SUCH THAT:
1) THERE IS EXACTLY ONE <WFS,1> NAMED
<WFS NAME,1> IN THE COLLECTION WHOSE
SCANNER <$S,1> = $S<SA,1>'<WFS NAME,1>'
HAS ATTRIBUTE <SA,1> = N | S | F | U .
2) NO PAIR <WFS,1> AND <WFS,2>
IN THE COLLECTION HAVE SCANNERS
<$S,1> = $S<SA,1>'<WFS NAME,1>'
AND
<$S,2> = $S<SA,2>'<WFS NAME,2>'
WITH <WFS NAME,1> = <WFS NAME,2> .

DENOTE THIS SET BY <STATE>
AND ITS ELEMENTS BY <STATE,I> ;

```
@IS_WFS = <WFS>  ;
@IS_BLOCK = <BLOCK>  ;
@IS_CONSTITUENT = <CONSTITUENT>  ;
@IS_TYPE =<TYPE>  ;
@IS_DIRECTION = L | R  ;
@IS_BITS = <BITSTRING>  ;
@IS_CHRS = <CHARACTERSTRING>  ;
@IS_NUMB = <NUMBER>  ;
@IS_NAME = <WFS NAME>  ;
```

@ MAPS BASIC FUNCTIONS INTO THE SET OF FUNCTIONS
JAM X JAM X...X JAM ---> JAM
WHERE SOME OF THEM ARE DEFINED AS FOLLOWS:

```
@EXEC(<STATE,1>) = <WFS NAME,1> OF THE SCANNER <$S,1>
                   CONTAINED IN THE UNIQUE <WFS,1>
                   IN <STATE,1> SUCH THAT
                    <SA,1> = F | N | S | U  ;
@CONTENT(<WFS NAME,1>,<STATE,1>) = THE UNIQUE <WFS,1> IN
                   <STATE,1> WHICH CONTAINES THE SCANNER
                   <$S,1>=$S<SA,1>'<WFS NAME,1>' ;
```

```
@KILL(<WFS NAME,1>,<STATE,1>) = <STATE,1> -
                    @CONTENT(<WFS NAME,1>,<STATE,1>) ;
@CREATE(<WFS NAME,2>,<WFS,1>,<STATE,1>) = <STATE,1> +
                                                <WFS,2>
                    WHERE <WFS,2> IS IDENTICAL TO <WFS,1>
                    EXCEPT THE <WFS NAME,1> IN THE SCANNER
                    OF <WFS,1> IS REPLACED BY <WFS NAME,2> ;
    /* - AND + ABOVE DENOTE THE SET THEORETICAL OPERATIONS
       OF DIFFERENCE AND UNION */
@BLOCK_AT(R,<$(,1><BLOCK*?,1><$S,1><BLOCK,1><BLOCK*?,2>
                    <$),1> ) = <BLOCK,1> ;
@BLOCK_AT(R,<$S,1><BLOCK,1>) = @BLOCK_AT(L,<$S,1><BLOCK,1>)
                    = <BLOCK,1>  ;
@CONSTITUENT_AT(R, ...<$S,1><CONSTITUENT,1>... )
                    = <CONSTITUENT,1>  ;
@DELETE(R,<$(,1><BLOCK*?,1><$S,1><BLOCK,2><BLOCK*?,3><$),1>)
         = <<$(,1><BLOCK*?,1><$S,1><BLOCK*?,3><$),1>  ;
@INSERT(R,<BLOCK,2>,
           <$(,1><BLOCK*?,1><$S,1><BLOCK*?,3><$),1>)
         = <$(,1><BLOCK*?,1><$S,1><BLOCK,2><BLOCK*?,3><$),1>;
@INBLD(R,...<$(,1><ORDINARY CONSTITUENT*?,1><$S,1>
                    <ORDINARY CONSTITUENT*?,2><$),1> ... )
      = ... <$(,1><ORDINARY CONSTITUENT*?,1>
                    <ORDINARY CONSTITUENT*?,2><$S,1><),1> ...;
@SHIFT(R, ...<$S,1><CONSTITUENT,1> ...)
             = ... <CONSTITUENT,1><$S,1> ... ;
@RESTORE(<WFS,1>) = <$S,1><BLOCK,1>  ;

@SCANNER_ATTR_OF( ... $S<SA,1>'<WFS NAME,1>' ...) = <SA,1> ;
@SET_SCANNER_ATTR(<SA,1>, ... $S<SA,2>'<WFS NAME,2>' ...)
                          = ... $S<SA,1>'<WFS NAME,2>' ... ;
@TYPE_OF($<TYPE,1> ... ) = <TYPE,1>  ;
@BITS_IN($B<BA,1>'<BITSTRING,1>') = <BITSTRING,1>  ;
@CHRS_IN($C<CA,1>'<CHARACTERSTRING,1>')
                  = <CHARACTERSTRING,1>  ;
@NUMB_IN($D<DA,1>'<NUMBER,1>') = <NUMBER,1>  ;
@NAME_IN($R<RA,1>'WFS NAME,1>') = <WFS NAME,1> ;



@ MAPS CONSTANTS INTO JAM  AS FOLLOWS:

@$B = B  ;
@$C = C  ;
@$D = D  ;
@$P = P  ;
@$R = R  ;
@$LEFT_PAREN = (   ;
@$RIGHT_PAREN = )   ;
@LEFT = L   ;
@RIGHT = R   ;
```

NEXT @ IS EXTENDED, BY NATURAL HOMOMORPHISM, ONTO THE SET OF TERMS INDUCTIVELY AS FOLLOWS:

IF V IS A VARIABLE, THEN @V IS THE IDENTITY MAPPING F(V)=V OF JAM ONTO ITSELF ;

IF C IS A CONSTANT , @C IS DEFINED ABOVE ;
IF T IS A TERM OF THE FORM

F(T1,...,TM)

WHERE F IS A BASIC FUNCTION AND T1,...,TM ARE TERMS, THEN @T IS THE MAPPING

@F(@T1,...,@TM)

FROM JAM X JAM X...X JAM INTO JAM ;

LASTLY @ IS EXTENDED ONTO THE SET OF FORMULAS INDUCTIVELY AS FOLLOWS:

IF A IS A FORMULA OF THE FORM

T1=T2

WHERE T1 AND T2 ARE TERMS , THEN @A IS THE PROPOSITIONAL FUNCTION

@T1 = @T2

WHERE THE ' = ' IS A WEAK EQUALITY RELATION (SEE 2.2.) DEFINED ON JAM ;

IF A IS A FORMULA OF THE FORM

P(T1,...,TM)

WHERE P IS A BASIC PREDICATE AND T1,...,TM ARE TERMS , THEN @A IS THE PROPOSITIONAL FUNCTION

@P(@T1,...,@TM) ;

FOR OTHER FORMULAS THE FOLLOWING IDENTITIES HOLD:

@(FORM1 | FORM2) = @FORM1 | @FORM2 ;
@(FORM1 & FORM2) = @FORM1 | @FORM2 ;
@(FORM1 => FORM2) = @FORM1 => @FORM2 ;
@(¬FORM) = ¬ @FORM ;

4. THE NUCLEUS MACRO PROCESSOR NUCMAC

4.1. MACRO SYNTAX

```
<MACRO INSTRUCTION> ::= $CM'MNEMONIC>' <BLOCK*?>
<MACRO DEFINITION> ::= $(XM <MACRO INSTRUCTION PROTOTYPE>
                            <BLOCK> $)XM
<MACRO INSTRUCTION PROTOTYPE> ::= <$(> $CM'<MNEMONIC>'
                                   <$P*?> <$)>
<MNEMONIC> ::= <CHARACTERSTRING>
```

4.2. NUCMAC

```
/*
   NUCMAC IS A SIMPLE MACRO PROCESSOR. IT REPLACES A MACRO
CALL BY THE MACRO BODY OF THE CORRESPONDING MACRO DEFINITION
WITH REPLACEMENT OF FORMAL PARAMETERS FOR ACTUAL ONES.
A MACRO DEFINITION MAY IN TURN CONTAIN A MACRO CALL,
BUT SINCE THERE IS NO CONDITIONAL EXPANSION FEATURE, NUCMAC
CANNOT HANDLE RECURSIVE MACROS.
   ACTUAL PARAMETERS MUST BE SINGLE BLOCKS. IF A FORMAL
PARAMETER SPECIFIES A TYPE AS ITS ATTRIBUTE, THEN THE
CORRESPONDING ACTUAL PARAMETER MUST BE A SINGLE CONSTITUENT
OF THIS TYPE.
   NUCMAC CONSISTS OF SEVEN WFS', APPROXIMATELY 100
INSTRUCTIONS, AND APPROXIMATELY 500 CONSTITUENTS. SPLITTING
EVEN A SHORT NUCLEUL PROGRAM INTO A NUMBER OF WFS'S MAKES
PROGRAMMING EASIER, DOCUMENTATION MORE LEGIBLE, AND FUTURE
CHANGES EASIER.
   THE SEVEN WFS'S OF NUCMAC ARE:

MACPRINI :THIS IS THE WFS THROUGH WHICH NUCMAC IS ENTERED
          AND INITIALIZED. AFTERWORDS CONTROL IS GIVEN TO
MACPRCTR :WHICH SCANS THE PROGRAM TO BE EXPANDED, AND WHEN
          IT FINDS A MACRO CALL TAKES APPROPRIATE ACTION.
          FIRST IT CALLS THE SHORT ROUTINE
IND      :WHICH CHECKS WHETHER THERE IS A CORRESPONDING
          MACRO DEFINITION, AND IF SO MAKES A COPY OF IT ON
          THE WFS
MACRO    : AND THEN GIVES CONTROL TO
MACPRCAL :WHICH BUILDS A LIST OF CORRESPONDING FORMAL AND
          ACTUAL PARAMETERS ON THE WFS
PARAM    :AND INCLUDES THE MACRO BODY IN THE PROGRAM, AND
          RETURNS CONTROL TO MACPRCTR.
          FINALLY, THERE IS ASSUMED TO BE A WFS
MACERROR :NOT SPECIFIED HERE, TO WHICH CONTROL PASSES UNDER
          ALL ERROR CONDITIONS.

   THE FOLLOWING ASSUMPTIONS ARE MADE ON HOW NUCMAC
INTERACTS WITH ITS ENVIROMENT. THE PROGRAM TO BE EXPANDED
IS A WFS CALLED 'PROGRAM' . EACH MACRO DEFINITION IS A WFS
WHOSE NAME IS THE NAME OF THIS MACRO. NUCMAC RETURNS CONTROL
TO A WFS CALLED 'SYS.' .

   THE PROGRAM LISTING WHICH FOLLOWS IS ALSO INTENDED TO
SERVE AS AN EXAMPLE OF A NUCLEUL PROGRAM, AND HENCE IT IS
EXTENSIVELY ANNOTATED. COMMENTS ARE ENCLOSED BETWEEN THE
CHARACTER PAIRS '/*' AND '*/' , E.G. AS IN
/* THIS IS A COMMENT */ .
*/
```

```
$SN'MACROINI'

/* ASSIGNING TO THIS SCANNER THE ATTRIBUTE 'N'
MAKES IT AN EXECUTION SCANNER, AND IS EQUIVALENT TO
DECLARING ITS WFS TO BE THE MAIN PROCEDURE */

$(X'MACPRINI'

$CK'COPY' $( $( $) $) $RN'PARAM'

/* WFS PARAM IS OPENED CONSISTING OF EXTERNAL PAIR OF
PARANTESES WHICH AUTOMATICLY ASSUME THE ATTRIBUTE 'X',
AND AN INTERNAL BLOCK IS INITIALIZED.
PARAM WILL BE USED IN A STACK FASION TO BUILD AND STORE
LISTS OF CORRESPONDING FORMAL AND ACTUAL PARAMETERS. EACH
SUCH LIST IS ENCLOSED IN PARANTESES AND THEREFORE CONSTITUTE
A BLOCK.LIST OF ACTUALLY EXPANDED MACRO IS ALWAYS THE
RIGHTMOST, IE. THE TOP ONE ON A STACK. THIS STACK SETUP
ALLOWS FOR HANDLING OF NESTED MACRO CALLS. THE ALREADY
INITIALIZED EMPTY BLOCK SERVES AS A BOTTOM OF THE STACK. */

$CK'SHFT' $RR'PARAM'
$CK'SHFT' $RR'PARAM'

/* SCANNER POSITION WAS INITIALISED BY SHIFTING IT INSIDE
THE INTERNAL BLOCK FROM OUTSIDE OF THE WFS PARAM. */

$P 'MACPROTR'
/* CONTROL IS GIVEN TO MACPROTR WHOSE SCANNER IS ASSUMED TO
BE IN A RESTORED POSITION */

$)X'MACPRINI'



$(X'MACPROTR'

/* THIS IS THE CENTRAL SCANNING LOOP. CONSTITUENT AFTER
CONSTITUENT IS CHECKED ON THE WFS  PROGRAM  AND APPROPRIATE
ACTION IS TAKEN. WHEN THE END OF  PROGRAM  IS REACHED,
CONTROL IS GIVEN TO THE SYSTEM */
/* SINCE THE ACTION IN CASE OF AN ERROR IN THE EXPANDED
PROGRAM IS NOT SPECIFIED YET, CONTROL IS GIVEN TO  MACERROR
IN THIS CASE */

/* MACRO CALL ? */
/* CONSTITUENT WILL BE TESTED FOR TYPE 'C' AND ATTRIBUTE
'M' DENOTING MACRO INSTRUCTION KEY WORD */

$CK'TEST' $RR'PROGRAM' $C 'T= ' $C 'C'
$(S
$CK'TEST' $RR'PROGRAM' $C 'A= ' $C 'M'
```

```
$(S     /* YES, MACRO CALL */
/* BUILD AN INSTRUCTION TO COPY THIS MACRO ONTO WFS  MACRO.
TO DO THAT CONVERT THE MACRO NAME INTO REFERENCE CONSTITUENT
AND INSERT IT INTO A SKELETON INSTRUCTION ON WFS  IND .
THIS TRICK IS USED IN PLACE OF INDIRECT REFERENCE,WHICH
WE DO NOT HAVE */

$CK'RSTR' $R 'MACPRCAL'

/* SCANNER WAS RESTORED TO THE POSITION OUTSIDE THE WFS
MACPRCAL. NOTICE THAT NO DIRECTION WAS SPECIFIED IN THIS
CASE */

$CK'RSTR' $RL'IND'

/* SCANNER WAS RESTORED TO THE LEFT PARENTESIS OF THE BLOCK
IN WHICH IT RESIDED ON THE WFS IND .  NOTICE THAT DIRECTION
'L' FOR LEFT WAS SPECIFIED IN THIS CASE. */

$CK'SHFT' $RR'IND'
$CK'MOVE' $RR'IND' $R 'SYS.FSL'

/* SCANNER WAS POSITIONED AT THE DUMMY FIRST ARGUMENT OF
THE SKELETON INSTRUCTION, AND THE ARGUMENT WAS CLEARED. */

$CK'COPY' $RR'PROGRAM' $RR'IND'
$CK'CVRT' $C 'R/ ' $RR'IND'

/* MACRO NAME WAS INSERTED AND CONVERTED INTO REFERENCE
TYPE CONSTITUENT THUS MAKING IT THE ACTUAL FIRST PARAMETER
OF THE COPY INSTRUCTION TO BE EXECUTED LATER. */
$CK'SHFT' $RL'IND'

/* SCANNER WAS POSITIONED BEFORE THE INSTRUCTION */
$R 'IND'

/* CONTROL WAS GIVEN TO THE NEWLY CREATED INSTRUCTION
ON THE WFS IND */

$)N
$)FU

/* GET HERE IF IT WAS NOT MACRO CALL */

/* FORMAL PARAMETER TO BE SUBSTITUTED ? */

$CK'TEST' $RR'PROGRAM' $C 'T= ' $C 'P'
$(S     /* YES, PARAMETER LIST RESET */
$CK'RSTR' $RL'PARAM'
$(N     /* BEGIN OF PARAMETER SEARCH LOOP */
$CK'MOVE' $RR'PARAM' $RL'PARAM'
```

```
/* PARAMETER LIST CONSISTS OF CONSECUTIVE PAIRS OF
CORRESPONDING ACTUAL AND FORMAL PARAMETERS.  SCANNER WAS
JUST POSITIONED BETWEEN THE ACTUAL ,ON THE LEFT, AND
FORMAL, ON THE RIGHT ,PARAMETERS.    */

$CK'TEST' $RR'PROGRAM' $C ' = ' $RR'PARAM'

/* CORRESPONDENCE BETWEEN FORMAL PARAMETERS IN MACRO BODY
AND IN PARAMETER LIST WAS CHECKED */

$(FL       /* NOT THIS PARAMETER YET */
$CK'SHFT' $RR'PARAM'
/* POSITION SCANNER AT THE NEXT PAIR OF PARAMETERS */
$)N

/*  IF PARAMETER NOT FOUND GO BACK TO THE BEGINNING OF THE
SEARCH LOOP. DO NOT CROSS THIS SUCCESFUL PARENTESIS
BECAUSE YOU ARE NOW IN NEUTRAL STATE */

$)S      /* GO OUTSIDE THROUGH THIS PARENT WHEN PARAMETER
            FOUND */

$CK'MOVE' $RR'PROGRAM' $R 'SYS.FSL'
$CK'COPY' $RL'PARAM' $RL'PROGRAM'
/* PARAMETER SUBSTITUTION WAS JUST MADE */

$CK'RSTR' $R 'MACPRCTR'
/*  END OF THE BRANCH. RESTORE THE SCANNER TO THE BEGINNING
OF THE CENTRAL SCANNING LOOP. */

$)N

/* END OF MACRO BODY MARKER ? */

$CK'TEST' $RR'PROGRAM' $C ' = ' $CS'ENDBODY'
/* THE SYSTEM MARKER 'ENDBODY' IS PLACED AT THE END OF
MACRO BODY WHEN MACRO BODY IS INCLUDED IN PROGRAM BY
MACPREAL */
$(S   /* WIND UP THIS MACRO EXPANSION */
$CK'MOVE' $RR'PROGRAM' $R 'SYS.FSL'
/* ENDBODY MARKER WAS REMOVED */

$CK'RSTR' $RL'PARAM'
$CK'SHFT' $RL'PARAM'
$CK'MOVE' $RR'PARAM' $R 'SYS.FSL'
/* THE TOP PARAMETER LIST IS DELETED FROM THE STACK*/

$CK'SHFT' $RL'PARAM'
/* THE LAST INSTRUCTION WAS TO REINSTITUTE PROCESSING OF
A PREVIOUS MACRO OR IF NONE ,PLACES THE SCANNER IN A FIRST,
DUMMY BLOCK */

$CK'PSTR' $R 'MACPRCTR'
```

```
/* END OF THIS BRANCH, RESTORE TO THE BEGINNING OF THE
CENTRAL SCANNING LOOP */

$)N

/* END OF PROGRAM TO BE EXPANDED ? */
$CK'TEST' $RR'PROGRAM' $C 'T= ' $C ')'
$(S
$CK'TEST' $RR'PROGRAM' $C 'A= ' $C 'X'
$(S
/*  RHIGHT PARENTISIS WITH AN ATTRIBUTE X WAS MET, IT
MEANS END OF PROGRAM */

$R 'SYS.'
/* CONTROL IS RETURN TO SYSTEM */

$)N
$)FL
/* GET HERE WHEN IT IS NOT THE END OF PROGRAM */

$CK'SHFT' $RR'PROGRAM'

/* LEAVE THIS CONSTITUENT ALONE AND START CHECKING THE
NEXT, IE. CLOSE THE CENTRAL SCANNING LOOP */

/* THIS RETURN TO THE BEGINNING OF THE CENTRAL SCANNING
LOOP IS DONE AUTOMATICLY SINCE THE EXTERNAL PARENTESIS,
THE ONE WITH THE ATTRIBUTE 'X' , CANNOT BE CROSSED */

$)X'MACPRCTR'



$(X'IND'
  /* THIS IS A MODYFIABLE PROCEDURE WHICH MAKES A COPY OF
     THE MACRO DEFINITION CORRESPONDING TO A MACRO CALL
     FOUND BY MACPRCTR ONTO THE WFS MACRO. THE FIRST
     ARGUMENT OF THE COPY INSTRUCTION BELOW IS SET BY
     MACPRCTR. IF NUCLEOL ALLOWED INDIRECT ADDRESSING, A
     MODYFIABLE PROCEDURE SUCH AS THIS WOULD NOT BE
     NECESSARY */
$SD'IND'
  /$ INTENDED INITIAL POSITION OF THE SCANNER IS INSIDE
     THE WFS */
  $CK'COPY' $R 'DUMMY' $R 'MACRO'
  $(W   /* NO MACRO WITH THIS NAME EXISTS */
    $R 'MACERROR'
  $)N
  $R 'MACPRCAL'
$)X'IND'
```

```
$(X'MACPRCAL'
/* INITIALISATION OF MACRO CALL */
/* MACRO WHICH IS CALLED WAS ALREADY COPIED ONTO WFS 'MACRO'
BY  'IND' */
$CK'RSTR' $RR'PARAM'
$CK'SHFT' $RR'PARAM'
$CK'RSTR' $R 'MACRO'
$CK'MOVE' $RR'MACRO' $RR'PARAM'
$CK'SHFT' $RR'PARAM'
$CK'TEST' $RR'PROGRAM' $C ' = ' $RR'PARAM'
$(FU      /* MACRO NAME NO GOOD */
$R 'MACERROR'
$)N
$(S       /* GENERATE PARAMETER LIST */
$CK'MOVE' $RR'PARAM' $R 'SYS.FSL'
$CK'MOVE' $RR'PROGRAM' $R 'SYS.FSL'
$N(     /* END OF PARAMETER LIST ? */
$CK'TEST' $RR'PARAM' $C 'T= ' $C ')'
$(FU      /* LOOP ON FORMAL PARAMETERS */
/* CHECK IF THE TYPE OF ACTUAL PARAMETER IS A DESIRED ONE */
$CK'TEST' $RR'PROGRAM' $C 'T=A' $RR'PARAM'
$(S      /* SET CORRESPONDING ACTUAL - FORMAL */
$CK'MOVE' $RR'PROGRAM' $RL'PARAM'
$CK'SHFT' $RR'PARAM'
$)N
$(FU      /* TYPES DO NOT MATCH */
$R 'MACERROR'
$)N
$)N
/* CLOSE THE LOOP TO NEXT PARAMETER */
$)S       /* EXIT HERE WHEN ALL FORMAL PARAMETERS PROCESSED */
/* SET MARKER 'ENDBODY' AND INCLUDE MACROBODY INTO THE
'PROGRAM', KILL 'MACRO' */
$CK'COPY' $CS'ENDBODY' $RR'PROGRAM'
$CK'RSTR' $RR'MACRO'
$CK'SHFT' $RR'MACRO'
$CK'MOVE' $RR'MACRO' $RR'PROGRAM'
$CK'MOVE' $R 'MACRO' $R 'SYS.FSL'
$CK'SHFT' $RR'PROGRAM'
/* CONTROL BACK TO THE CENTRAL SCANNING LOOP */
$R 'MACPRCTR'
$)N

$)X'MACPRCAL'
```

5. CONCLUSION

The main conclusion obtained from this research is that it is feasible and useful to combine the formal definition of a programming language with its implementation. Specifically, if this definition of NUCLEOL is augmented by programs which implement the basic functions and predicates listed in 2.1, it constitutes an interpreter for NUCLEOL. While these programs depend on a particular realization of NUCLEOL (host computer, data representation, etc.), a significant fraction of the interpreter (about half) is machine independent. Hence, as a by-product, we have considerably reduced the problem of transferring NUCLEOL from one computer to another.

The main task which was left undone is to use the system of axioms which relate the basic functions and predicates. While these axioms serve an intuitive purpose of explaining what particular basic functions and predicates are, they have not been used in the formal development. A challenging future development would be to prove that they are consistent and complete, the latter in the sense of defining NUCLEOL uniquely to a user.

REFERENCES

1. Naur, P., et al., "Revised Report on the Algorithmic Language ALGOL 60", Computer J. 5, 4, pp. 349-368, January, 1963.

2. McCarthy, J., "A Formal Description of a Subset of ALGOL", In Steel, T. B. (editor), "Formal Language Description Languages For Computer Programming", pp. 1-12, North Holland, 1966.

3. McCarthy, J. and Painter, J., "Correctness of a Compiler For Arithmetic Expressions", in "Mathematical Aspects of Computer Science", Proc. Symposia in Applied Math., Vol. 19, pp. 33-41, American Math Society, 1967.

4. Bandat, K., "On The Formal Definition of PL/1", Proc. AFIPS 1968 Spring Joint Computer Conference, Atlantic City, New Jersey, pp. 363-373, April 1968.

5. Sidlo, John R., "The List Processing Language NUCLEOL", Master of Science dissertation, University of Illinois, Department of Computer Science, Urbana, Illinois, August 1968.